PUCRS ESCOLA POLITÉCNICA

Algoritmos de Divisão e Conquista: Pesquisa Binária

O ALGORITMO:

- Seja $a_i, 1 \leq i \leq n$, uma lista de elementos

Example 3.6 Let us select the 14 entries

$$-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151$$

- Considere o problema de determinar se um determinado elemento o está presente na lista.
- Se x estiver presente, devemos determinar um valor j tal que $a_j = x$.
- Se x não estiver na lista, então j deve ser definido como zero.
- Deixe $P = (n, a_i, \dots, a_\ell, x)$ denotar uma instância arbitrária deste problema de pesquisa (n é o número de elementos na lista,$a_i, \dots, a_\ell$ é a lista de elementos, e x é o elemento procurado).
- Divisão e conquista pode ser usado para resolver esse problema.
- Seja $Small(P)$ verdadeiro se $n = 1$.
  - Nesse caso, $S(P)$ assumirá o valor i se $x = a_i$; caso contrário, assumirá o valor 0.
  - Então $T(1) = \theta(1)$.
- Se P tiver mais de um elemento, ele pode ser dividido (ou reduzido) em um novo subproblema como segue:
  - Escolha um índice q (no intervalo $[i, \ell]$) e compare x com a_q. Existem três possibilidades:
    1. $x = a_q$: Neste caso o problema P é resolvido imediatamente.
    2. $x < a_q$: Neste caso x deve ser procurado apenas na sublista a_i, $a_{i+1}, \dots, a_{q-1}$. Portanto, P se reduz a $P = (q - i, a_i, \dots, a_{q-1}, x)$.
    3. $x > a_q$: Neste caso a sublista pesquisada será a_i, $a_{q+1}, \dots, a_\ell$. E, P se reduz a $P = (\ell - q, a_{q+1}, \dots, a_\ell, x)$.

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i : l] of elements in nondecreasing
// order, 1 ≤ i ≤ l, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{
    if (l = i) then  // If Small(P)
    {
        if (x = a[i]) then return i;
        else return 0;
    }
    else
    { // Reduce P into a smaller subproblem.
        mid := ⌊(i + l)/2⌋;
        if (x = a[mid]) then return mid;
        else if (x < a[mid]) then
                return BinSrch(a, i, mid − 1, x);
            else return BinSrch(a, mid + 1, l, x);
    }
}
```

Algoritmos de Divisão e Conquista: Pesquisa Binária

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i : l] of elements in nondecreasing
// order, 1 ≤ i ≤ l, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{
    if (l = i) then  // If Small(P)
    {
        if (x = a[i]) then return i;
        else return 0;
    }
    else
    { // Reduce P into a smaller subproblem.
        mid := ⌊(i + l)/2⌋;
        if (x = a[mid]) then return mid;
        else if (x < a[mid]) then
                return BinSrch(a, i, mid − 1, x);
             else return BinSrch(a, mid + 1, l, x);
    }
}
```

Example 3.6 Let us select the 14 entries

$$-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151$$

- Neste exemplo, qualquer problema P é dividido (reduzido) em um novo subproblema. Esta divisão leva apenas $\theta(1)$ tempo.
- Depois de uma comparação com a_q, a instância restante a ser resolvida (se houver) pode ser resolvida usando novamente este esquema de dividir e conquistar.
- Se q é sempre escolhido tal que a_q é o elemento do meio (ou seja, $q = \lfloor (n+1)/2 \rfloor$), então o algoritmo de pesquisa resultante é conhecido como pesquisa binária.

- Observe que a resposta para o novo subproblema também é a resposta ao problema original P. Uma vez que achamos o índice correto chegamos a resposta global!!!
- Não há necessidade de nenhuma combinação.
- O Algoritmo $BinSrch$ descreve esse método de busca binária, e possui quatro entradas $a[\ \ ], i, l, x$.
- É inicialmente invocado como $BinSrch(a, 1, n, x)$.

PUCRS ESCOLA POLITÉCNICA

Algoritmos de Divisão e Conquista: Pesquisa Binária

ANALISANDO O ALGORITMO:

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i : l] of elements in nondecreasing
// order, 1 ≤ i ≤ l, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{
    if (l = i) then  // If Small(P)
    {
        if (x = a[i]) then return i;
        else return 0;
    }
    else
    { // Reduce P into a smaller subproblem.
        mid := ⌊(i + l)/2⌋;
        if (x = a[mid]) then return mid;
        else if (x < a[mid]) then
                 return BinSrch(a, i, mid − 1, x);
             else return BinSrch(a, mid + 1, l, x);
    }
}
```

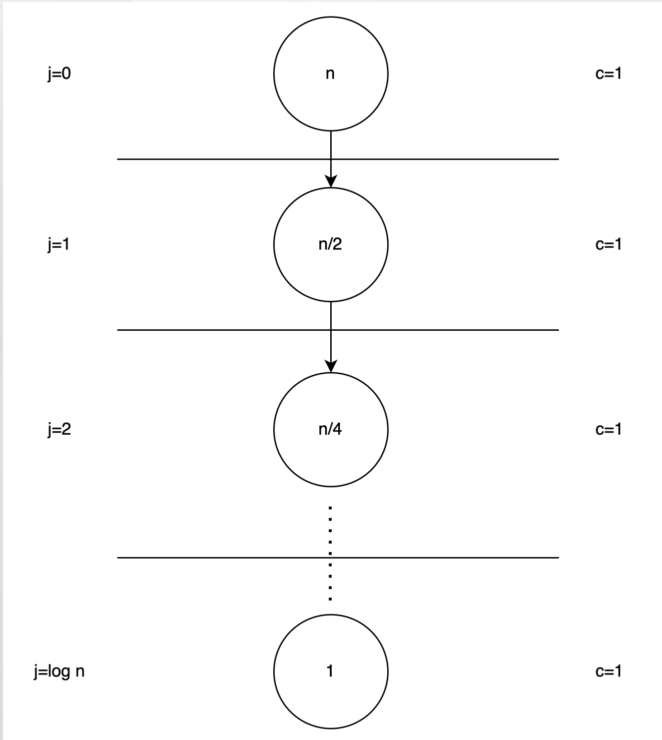
A.16: O algoritmo de pesquisa binária funciona de forma correta.

- **Prova (informal).** O algoritmo funciona para quando $n = 1$, pois ou retorna o índice correto ou retorna zero.
- Se $n > 1$ pergunta se o valor do meio indicado pelo índice q é igual a $x = a_q$, caso positivo retorna o índice. Caso contrário, reduz o problema e chama recursivamente,
- Como o problema fica menor a cada chamada recursiva garantimos que o algoritmo termina ou com o índice correto ou com o valor zero.

- Portanto, podemos descrever a Pesquisa Binária com a seguinte recorrência:

A.17: Para alguma constante c

$$T(n) = \begin{cases} T(1) & se \quad n = 1 \\ T(n/2) + c & se \; n > 1 \end{cases}$$



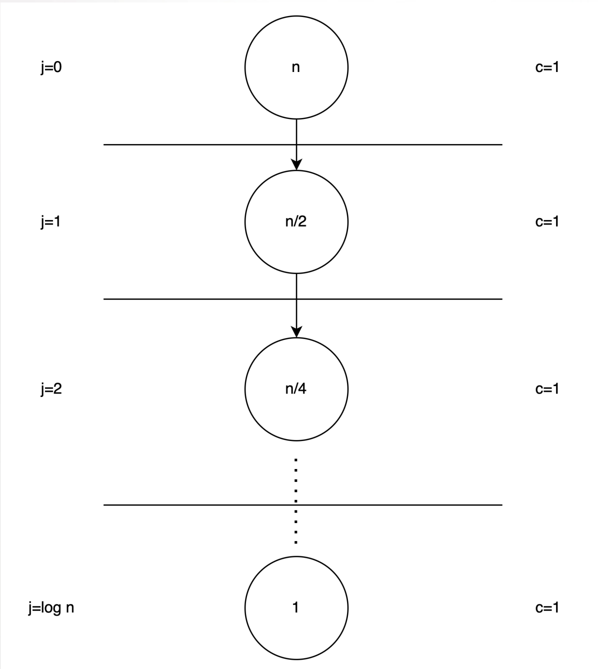
PUCRS ESCOLA POLITÉCNICA

Algoritmos de Divisão e Conquista: Pesquisa Binária

ANALISANDO O ALGORITMO:

A.17: Para alguma constante c

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{T(1)} & \boldsymbol{se\ \ n = 1} \\ \boldsymbol{T(n/2) + c} & \boldsymbol{se\ n > 1} \end{cases}$$



$$\sum_{j=0}^{log_2 n} c = log_2 n \cdot c$$

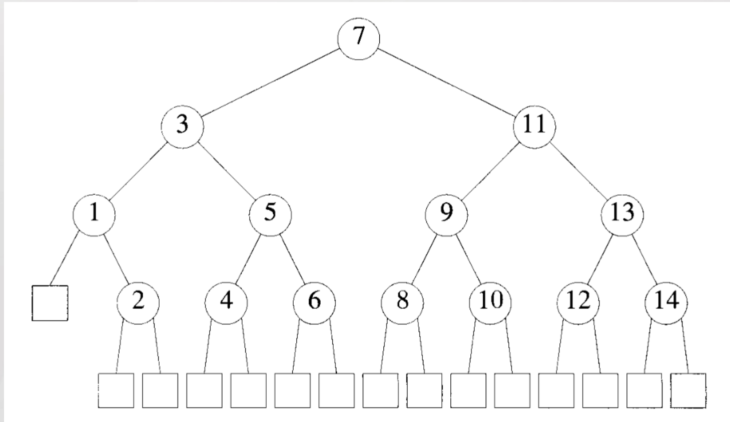
$$O(log_2 n \cdot c) = O(log_2 n)$$

- Portanto, no pior caso teremos $log_2 n$ chamadas recursivas que no fundo da recursão custam $c = 1$ e o custo para dividir é c e o custo para combinar é zero.
- Portanto, no pior caso temos que

$$T(n) \text{ é } O(log_2 n)$$

Example 3.6 Let us select the 14 entries

$$-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151$$



$x = 9$	*low*	*high*	*mid*
	1	14	7
	1	6	3
	4	6	5
			found

a:	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
Elements:	−15	−6	0	7	9	23	54	82	101	112	125	131	142	151
Comparisons:	3	4	2	4	3	4	1	4	3	4	2	4	3	4

ANALISANDO O ALGORITMO:

A.17: Para alguma constante c

$$\boldsymbol{T(n)} = \begin{cases} \boldsymbol{T(1)} & \boldsymbol{se \;\; n = 1} \\ \boldsymbol{T(n/2) + c} & \boldsymbol{se\, n > 1} \end{cases}$$

Pelo método mestre:

- $T(n) = aT(n/b) + f(n) = T(n/2) + 1$

onde $a = 1, b = 2, f(n) = 1$

Caso 2:

- Então temos que $n^{log_b a} = n^{log_2 1} = n^0 = \Theta(1)$
- Como $f(n) = O\left(n^{\log_b a}\right)$, pois $n^{\log_b a} = n^{log_2 1} = 1$.
- Portanto, $n^{log_b a} = f(n)$, a solução é $T(n) = \Theta(f(n) \log_2 n) = \Theta(1 \log_2 n) = \Theta(\log_2 n)$.